VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2020 Ondřej Slíva

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ondřej Slíva**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: BiddingTools s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Dohnálek, Ph.D.**

Konzultant bakalářské práce: Bc. Jiří Kostov

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

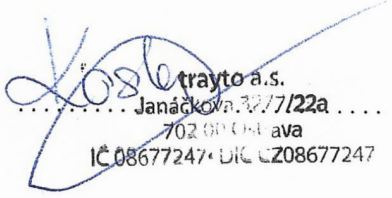
prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 13. dubna 2020 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 13. dubna 2020



Rád bych na tomto místě poděkoval společnosti trayto a.s. za umožnění praxe, a jejím zaměstnancům, kteří mi byli vždy ochotni pomoci. Dále bych chtěl poděkovat mému vedoucímu Ing. Pavlu Dohnálkovi, Ph.D., který mi poskytl pomoc a rady při zpracování této práce.

Abstrakt

Hlavním cílem této práce je seznámit čtenáře s průběhem mé odborné praxe ve firmě trayto a.s., kde pracuji jako Software Developer. Během praxe bylo mým hlavním cílem vytvořit nástroj, který pomůže s párováním produktů na zbožových srovnávačích, aby tato činnost již nemusela být prováděna manuálně. Tento nástroj lze použít pro srovnávače Heureka i pro Zboží.cz. Nástroj slouží ke stahování informací o produktech e-shopu, a těm nespárovaným vyhledává vhodné produktové karty pro napárování. Navrhované produktové karty jsou pak k dispozici uživateli, jenž je zkontroluje a rozhodne o jejich správnosti. E-shop poté obdrží soubor s upravenými produkty, což zajistí jejich spárování. V době odevzdání této bakalářské práce je dokončena implementace pro Heureka.cz a Heureka.sk. Implementace pro srovnávač Zboží.cz zatím nebyla dokončena. Díky tomuto nástroji firma poskytuje další službu. Ta, spolu s ostatními, pomáhá e-shopům zvýšit obrat na zbožových srovnávačích.

Klíčová slova: Odborná praxe, PHP, JavaScript, API, webová aplikace, zbožové srovnávače

Abstract

The main goal of this work is to acquaint the reader with the course of my professional practice in the company trayto, where I work as a Software Developer. During my practice, my main goal was to develop a tool that will help with the pairing of products on goods comparators, so that this activity no longer has to be performed manually. This tool can be used for comparator Heureka and for Zboží.cz. The tool is used to download information about e-shop products and to search for suitable product cards for the unpaired products. The proposed product cards are then available to the user, who checks them and decides on their correctness. The e-shop will then receive a file with the modified products, which will ensure their pairing. At the time of submitting this bachelor work, the implementation for Heureka.cz and Heureka.sk has been completed. The implementation for the comparator Zboží.cz has not been completed yet. Thanks to this tool, the company provides another service. Together with others, it helps e-shops to increase turnover on goods comparators.

Keywords: Professional practice, PHP, JavaScript, API, web application, goods comparators

Obsah

Seznam použitých zkratek a symbolů

EAN	–	European Article Number
PHP	–	Hypertext Preprocessor, originally Personal Home Page
SQL	–	Structured Query Language
JS	–	JavaScript
HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets
XML	–	Extensible Markup Language
API	–	Application Programming Interface
JSON	–	JavaScript Object Notation
URL	–	Uniform Resource Locator
HTTP	–	Hypertext Transfer Protocol
URI	–	Uniform Resource Identifier
AJAX	–	Asynchronous JavaScript and XML
ID	–	Identification

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době najít uplatnění v oblasti informatiky není velký problém. Firmy hledají IT specialisty všech zaměření, tudíž nabídka pracovních míst je velmi rozsáhlá. I přesto však požaduje stále více firem určité praktické zkušenosti. Naproti tomu je výuka na vysokých školách zaměřena spíše na teoretické znalosti než na ty praktické.

Kvůli těmto důvodům jsem již během studia docházel do firmy BiddingTools Group s.r.o., kde jsem postupně zjišťoval, jak práce v IT oboru skutečně vypadá. Naučil jsem se, jak probíhá vývoj softwaru, práce v týmu, ale také mnoho dalšího. Získal jsem zde velké množství zkušeností, které jsou velmi cenné pro mé budoucí povolání.

Během praxe byl mým hlavním úkolem vývoj nástroje, jenž se nazývá Jack Párovač. Tento nástroj má za cíl pomoci s párováním produktů na zbožových srovnávačích. Firma trayto a.s. vytváří nástroje, které mají za cíl pomoci e-shopům na zbožových srovnávačích. Jsou to nástroje, které slouží pro bidování, upravují feedy, nebo porovnávají ceny s konkurencí. To, co společnosti chybělo, byl nástroj, jenž by e-shopům pomohl spárovat jejich produkty na zbožových srovnávačích. A přesně tuto mezeru by měl vyplnit nástroj Jack Párovač.

Na tomto úkolu jsem pracoval samostatně a vyzkoušel jsem si tak všechny činnosti, které se pojí s vývojem softwaru, což mi pomohlo získat širší pohled na celý vývojový proces. Tato zkušenost přinesla zřetelnější perspektivu budoucího směrování a zaměření mého kariérního růstu.

Nejdříve představím společnost, ve které pracuji, a popíši své pracovní zázemí. Následující kapitola se krátce zabývá technologiemi, jež byly na vývoj použity. Po ní následují hlavní kapitoly. V první z nich popisuji vyvíjenou aplikaci a její hlavní funkce. V té následující popisuji úkoly, jež jsem během své praxe řešil, a detailněji popisuji ty obsáhlejší z nich. V další kapitole uvádím uplatněné znalosti, a znalosti které mi chyběly. Následuje závěr, ve kterém absolvovanou praxi hodnotím.

2 Popis firmy a pracovní náplň studenta

Firma Biddingtools Group s.r.o. je poměrně malá společnost, která se nachází v Ostravě. Spíše než o IT firmu se jedná o firmu marketingovou, jež vyvíjí své vlastní nástroje, které zajišťují pomoc e-shopům na zbožových srovnávačích, jako je Heureka nebo Zboží.cz. [1]

2.1 Vznik společnosti trayto a.s.

Během mého působení v této společnosti došlo ke sloučení, tedy firma se stala součástí větší skupiny ette capital a.s. Ta má pod sebou více firem, jež se zabývají informačními technologiemi i například udržitelností životního prostředí. [2] Pro společnost Biddingtools Group, dnes již trayto a.s., to znamená, že v rámci této změny společnost, ve které jsem praxi vykonával, zanikla, a vznikla nová společnost se jménem trayto a.s. Firma v rámci tohoto spojení rozšířila portfolio projektů o další nástroje. Příkladem mohou být nástroje k poskytování finančních služeb. I nadále společnost vyvíjí marketingové projekty, jen pod jiným obchodním jménem.

2.2 Pracovní pozice

Svou praxi jsem absolvoval na pozici Software Developer. Vzhledem k tomu, že firma patří k menším, nemá pro každou činnost určeného specialistu, který ji vykonává, proto se každý vývojář může podílet na více činnostech. Rozdělení zde funguje na bázi projektů, kdy pro každý projekt existuje tým lidí. Týmy však nejsou zcela oddělené a navzájem spolu spolupracují, probírají různá řešení či sdílejí své zkušenosti. Pro mě to znamenalo, že kromě vývoje samostatného softwaru jsem řešení také analyzoval, navrhoval, testoval či nasazoval.

Mou hlavní pracovní náplní byl vývoj nástroje Jack Párovač. Na vývoji jsem pracoval převážně samostatně, kromě činností jako jsou code review, konzultace s kolegy ohledně implementace a podobně.

Kromě hlavního projektu jsem pracoval i na jiných, méně komplexních projektech, které se již aktivně nevyvíjejí, ale občas je bylo potřeba rozšířit či upravit. Příkladem takového projektu je projekt Cenotvorby, jenž analyzuje ceny konkurence na základě klientských požadavků.

2.3 Postup vývoje

Celý projekt byl vyvíjen agilně, a to pomocí metody Scrum. Jedná se o iterativní a inkrementální agilní metodiku řízení vývoje softwaru. Klíčovým principem na scrumu je, že během vývoje projektu, se požadavky mohou často měnit. Na rozdíl od "tradičního sekvenčního přístupu" umožňuje lepší reakci na nové požadavky a jejich rychlé dodání. [3] Pravidelně jsem konzultoval dosažený postup a spolu s ostatními kolegy určoval následující činnosti.

3 Použité technologie

Aplikace je rozdělena na dvě části: backend a frontend. Na backendu probíhá veškerá logika, která je pro běh aplikace potřebná. Jsou zde umístěny veškeré algoritmy, komunikace s databází a podobně. Backend poskytuje API rozhraní, díky kterému je možná komunikace s frontendem. Frontend tedy posílá požadavky a backend na ně odpovídá.

3.1 Backend

Backend aplikace je psán v programovacím jazyce PHP 7. Tento jazyk byl zvolen převážně proto, že se jedná o hlavní programovací jazyk používaný ve firmě. Vzhledem k tomu, že trayto je poměrně malá firma, tak by nedávalo smysl, kdyby byl každý nástroj psán v jiném jazyce. Dále v projektu využívám některé interní knihovny, které jsou taktéž napsány v jazyce PHP.

Jako datové úložiště jsme zvolili SQL databázi MySQL. Je to pravděpodobně nejčastěji používaná kombinace ve spojení s Apachem. Některé operace probíhají vždy při ukládání či mazání z databáze. Kvůli tomu bylo pro jejich implementaci vhodné použití databázových procedur a triggerů.

Na backendu je dále použit microframework Slim 4, který umožnuje velmi jednoduše napsat webové API. Poskytuje router, který mapuje volání HTTP metod a URI na konkrétní metody. Podporuje také parametry, rozpoznání vzorů v URL, použití middlewaru, nebo také dependency injection pro snadné předávání závislostí. [4]

Na backendovou část aplikace jsou napsány testy. K tomuto je použita knihovna pro PHP, která se nazývá Codeception. Codeception je jedna z nejpoužívanějších knihoven, jež se používá pro testování v jazyce PHP. Umožňuje vytvořit unit testy, integrační testy, ale také například API testy, pomocí kterých se dá testovat API, jako by byly posílány reálné požadavky. Codeception poskytuje také jednoduché možnosti, jak při testech používat demo data, takže je možné testovat stejně, jako bychom se dotazovali na API. [5]

3.2 Frontend

Jako rozhraní pro uživatele slouží webové stránky. Aplikace je tak přístupná kdekoliv, kde je k dispozici internetové připojení. Stránky se opět generují v jazyce PHP, jen vrací kód v jazyce HTML. Stránky jsou stylovány pomocí CSS.

Jelikož jazyk PHP běží na straně serveru, neumožňuje dynamickou práci s webovými stránkami. Pokud cokoliv chceme provést, je nutné poslat požadavek na server, který nám opět pošle novou odpověď a celá stránka se musí překreslit. Toto řeší jazyk JavaScript, jenž funguje na straně klienta. Umožňuje tedy uživatelsky přívětivější práci s webovým rozhraním. Za pomoci něho lze měnit obsah stránky za běhu, tvořit dynamické menu, animace a podobně.

Pro JavaScript existuje knihovna jQuery. Ačkoliv období největší popularity již má nejspíš za sebou a v dnešní době se místo ní používá Rect, Angular či jiné JavaScriptové frameworky,

je stále poměrně rozšířená. Je to především proto, že má jednoduchou syntax, zjednodušuje manipulaci s obsahem stránky, ale také poskytuje mnoho kompletních řešení, které by si jinak vývojáři museli napsat sami. Mimo jiné se v ní také mohou posílat požadavky AJAX, pomocí nichž lze komunikovat s backendovými API.

3.3 Ostatní

Některé technologie jsou použity na obou částech systému. Jedná se o verzovací systém Git. Git je jednoduchý, ale zároveň velice mocný systém pro správu verzí zdrojových kódů. Zaznamenává, jaké změny byly v kódu provedeny, kdy se to stalo, a kdo dané změny provedl. Změny je poté možno zpětně dohledat a v případě potřeby kdykoliv obnovit dřívější verzi. Současně může existovat více verzí kódu, takže na jednom projektu může pracovat více vývojářů současně.

Další technologií, která je použita v obou částech aplikace, je Composer. Jedná se o package manager pro jazyk PHP. Umožňuje nainstalovat knihovny, nebo také závislosti, a ty poté můžeme v daném projektu používat. Mimo jiné umí zjistit, zda existují nové verze, jestli knihovna má jiné závislosti, které v případě potřeby doinstaluje a mnoho dalšího. Pro své fungování používá dva soubory - composer.json, ve kterém je uvedeno, které knihovny má použít, a composer.lock, ve kterém jsou jejich konkrétní verze, včetně jejich závislostí. Na základě těchto dvou souborů je možno všechny závislosti stáhnout, takže každý vývojář má všechny potřebné knihovny k dispozici.

4 Vyvíjená aplikace

V rámci mé praxe jsem pracoval převážně na jednom projektu. Byl to vývoj nástroje Jack Párovač, který se zabývá párováním produktů na zbožových srovnávačích.

Párování produktů by se dalo popsat jako spojení produktu e-shopu se správnou produktovou kartou na zbožovém srovnávači. Pokud dojde ke spárování produktu e-shopu a produktové karty srovnávače, zobrazí se nabídka daného e-shopu na produktové kartě srovnávače. Toto párování probíhá pro každý produkt a ne vždy se podaří spárovat všechny produkty. Pokud e-shopy, které inzerují na zbožových srovnávačích, chtějí srovnávače skutečně využít, měly by mít spárováno co nejvíce produktů.

4.1 Důvod vzniku aplikace

Nízká spárovanost produktů na zbožových srovnávačích trápí mnoho e-shopů, a to z toho důvodu, že nespárované produkty na srovnávači e-shopu generují minimální nebo žádný obrat.

Důvodů nízké spárovanosti e-shopu na srovnávačích může být hned několik. Většinou jde o nekvalitní data, která e-shop do srovnávače posílá prostřednictvím XML souboru. Typicky jde o chybějící informace v názvu produktu, špatná kategorizační cesta, nebo chybějící povinné parametry produktu. Dalším důvodem může být absence produktový karet ve srovnávači pro některý sortiment e-shopu.

Dalším faktem je, že řešit párování ručně je časově poměrně náročné. Je třeba najít správnou kartu, upravit produkt ve feedu tak, aby jí co nejlépe odpovídal, a toto je potřeba provést zvlášť pro každý produkt. Není třeba zdůrazňovat, že tímto způsobem může vzniknou také řada chyb. Díky těmto chybám se produkt nemusí vůbec spárovat, nebo může skončit napárovaný na úplně jinou produktovou kartu. Právě tyto problémy má pomoci řešit nástroj Jack Párovač.

4.2 Funkce aplikace

Aby aplikace mohla fungovat, bylo potřeba zprovoznit velké množství funkcionalit. Některé z nich se vztahují převážně k uživatelskému rozhraní. Mezi ně patří ty, jež jsou nezbytné pro párování produktů, jako například zobrazení produktů, nebo samostatné vytváření pravidel. Kromě toho rozhraní poskytuje mnoho dalších funkcionalit, jež sice nejsou přímo nezbytné pro párování, nicméně výrazně usnadňují práci. Příklady takových funkcionalit jsou filtrování nebo řazení produktů. Dále se jedná o drobnější úkoly, jako je přidání vizuálního označení, které vyjadřuje, zda již k produktu existuje pravidlo, nebo seznam již vytvořených pravidel.

Nejdůležitější, ale také nejobsáhlejší funkcí, zůstává párování produktů. Bez toho by aplikace postrádala smysl. Dalšími většími funkcionalitami jsou ty, které vidí klienti. Jedná se o analýzy pro nové klienty a měsíční reporty pro ty stávající.

4.2.1 Párování produktů

Hlavním cílem vyvíjené aplikace je pomoc e-shopům se spárovaností jejich produktů na zbožových srovnávačích. Aplikace vyhledává relevantní produktové karty, na které by mohl být daný produkt napárován. Poté tuto nabídku zobrazí uživateli. Ten rozhodne, zda je návrh správný, a chce produkt na danou kartu napárovat, nebo je návrh špatně, a nemá jej tedy již k párování nabízet.

Vzhledem k tomu, že na zbožových srovnávačích existuje velké množství produktových karet (například Heureka.cz uvádí, že jich má téměř 30 milionů [6]), a stále přibývají další, existuje mnoho takových, které jsou si velmi podobné. Nástroj proto není schopný přesně určit, jestli je daná produktová karta nejvhodnější, případně zda li je vůbec správná. Proto namísto automatického párování nabídne pouze možnosti, kam daný produkt může být napárován.

Rozhodnutí o správnosti těchto návrhů poté přenechá na uživateli. Ten poté návrhy projde a vyhodnotí, které jsou správné, a které nikoliv. Aplikace Jack Párovač na základě rozhodnutí uživatele vytvoří pravidla, jež mají být na produkt aplikována. Podle těchto pravidel se daný produkt upraví a spolu s ostatními produkty se opět pošle do srovnávače.

Data se do srovnávačů posílají ve formě XML souborů, tzv. feedů. Vzhledem k tomu, že ve společnosti je mimo jiné vyvíjen i nástroj XeMeL, který se zabývá úpravami XML feedů, tak přímo Jack Párovač přegenerování XML feedů neprovádí. Napojí se na API nástroje XeMeL a předá informace o tom, jaké hodnoty ve feedu nově mají být. I když zde probíhá spolupráce dvou nástrojů, navenek se Jack Párovač tváří jako jeden.

Výsledkem celého procesu je URL, na kterém je přegenerovaný XML soubor s upravenými hodnotami. Toto URL je poté vloženo do administrace srovnávače. Ten by měl změnu těchto produktů zaregistrovat a pokusit se produkt opět spárovat, tentokrát již úspěšně.

4.2.2 Proces párování na Heureka.cz

Logickým krokem pro párování produktů je použití EAN kódu. EAN je jedinečný identifikátor produktu. To ale není zcela pravda. EAN může být přidělen například 50 produktům a pro srovnávače se z nich stanou nerozpoznatelné varianty. Navíc chybovost v posílaných EAN kódech je poměrně velká, takže posílání EAN kódů pro většinu typů produktů není povinnost. [7]

Pro párování produktů se používá několik elementů. Jsou jimi SHOPITEM, PRODUCT, PRODUCTNAME, CATEGORYTEXT. [8]

– SHOPITEM - jedná se o jeden produkt e-shopu. V tomto elementu jsou obsaženy ostatní jako například CATEGORYTEXT, PRODUCTNAME a podobně. Pokud e-shop prodává například triko, které nabízí ve více variantách, pro každou variantu je potřeba posílat samostatný SHOPITEM. [9]

– CATEGORYTEXT - Kategorie produktu podle stromu aktivních kategorií na daném srovnávači. Může zde být použita i kategorie z e-shopu, u ní však párování není tak úspěšné. [9, 8]

– PRODUCTNAME - Jedná se o obecné pojmenování produktu, včetně označení výrobce. Většinou obsahuje produktové označení, případně i variantu produktu. [9]

– PRODUCT - Jedná se o rozšíření PRODUCTNAME. Obsahuje PRODUCTNAME + informaci navíc, která je platná pro konkrétní e-shop. Například prodloužená záruka, info o dopravě, distribuce produktu a podobně. [9]

Nejdříve srovnávač kontroluje kategorii daného produktu. Pokud produkt vůbec nemá kategorii, je vyhodnocen jako nespárovaný. Pokud je obsažena, existuje několik možností, kam se produkt zařadí. [8]

1. Ignorované kategorie - tento status je produktu přiřazen, pokud je jeho kategorie příliš krátká, nebo příliš obecná. Jinak řečeno, sem se řadí produkty, jejichž kategorii Heureka nerozpoznala a nebyla schopna namapovat na strom kategorií svého katalogu. [8]

2. Neaktivní kategorie - pokud má produkt neaktivní kategorii, znamená to, že pro něj již určitá kategorie existuje, ale zatím není spuštěna. [8]

3. Čekající kategorie - tento status je produktům přidělen, pokud robot nerozpozná jejich kategorii. Tyto produkty jsou přesunuty k ručnímu párování administrátorem. Pokud administrátor kategorie rozpozná, je produkt umístěn do kategorie s čekajícími produkty. [8]

Po vyhodnocení kategorie se přejde ke jménu produktu. Ten je obsažen v elementu PRODUCTNAME, nebo PRODUCT. Primárně probíhá párování na základě elementu PRODUCTNAME. Element PRODUCT se používá, pokud není PRODUCTNAME ve feedu obsažen, nebo nejsou dodrženy podmínky pro tyto elementy. Jestliže jsou splněny podmínky, srovnávač se snaží produkty spárovat na základě jejich názvu. Pokud je název správný, produkt se páruje, v opačném případě jde do nespárovaných produktů. [10]

Po tomto párovacím procesu se již produkt dále nepáruje a zůstane mu jeden ze statusů. Do procesu párování se opět dostane, pokud se například neaktivní kategorie stane aktivní, dojde k rozdělení kategorie, anebo také při změně některého ze zmíněných elementů. [8]

Z výše uvedeného vyplývá, že pro párování produktů jsou nejdůležitější dvě informace - CATEGORYTEXT a PRODUCTNAME/PRODUCT. Proto hlavním úkolem pro párování je nalezení správných hodnot pro tyto dva elementy.

4.2.3 Analýzy

Analýzu e-shop obdrží ve chvíli, kdy se rozhodne, že nástroj vyzkouší. Na stránkách společnosti navštíví sekci s nabídkou na párování, kde je možnost vygenerování analýzy. Vyplní zde své

přihlašovací údaje do srovnávače, email, telefon a (vzhledem k tomu, že na jednom účtu může být ve srovnávači více e-shopů) zvolí, pro které e-shopy chce analýzu vygenerovat. Nástroj zašle email o přijetí požadavku, z administrace zjistí URL feedu, informace o tom, které produkty jsou či nejsou spárované, a pokusí se zjistit, kolik produktů je schopen spárovat. Poté, co toto dokončí, odešle na obchodní oddělení společnosti e-mail, že může analýzu zkontrolovat, doplnit a informovat klienta.

Obchodní oddělení následně zašle klientovi analýzu. V ní poté potenciální klient nalezne, kolik produktů může být nově spárováno, jaká bude cena, a jaké další balíčky pro párování může využít. Na základě toho se rozhodne, zda některou z nabídek přijme, a zvolí si jednorázové či pravidelné párování, nebo nabídku odmítne. Analýzu je možné generovat opakovaně, podle potřeb e-shopu. Výsledky jednotlivých analýz e-shopu se mohou lišit, jelikož v e-shopech v čase dochází ke změnám sortimentu, například v důsledku sezónnosti.

4.2.4 Měsíční report

Měsíční report slouží pouze pro ty klienty, kteří se rozhodli pro dlouhodobou spolupráci. V těchto reportech lze najít vývoj počtu spárovaných a nespárovaných produktů za období spolupráce. Dále je zde vidět, kolik produktů za daný měsíc přibylo v XML feedu, kolik jich z feedu zmizelo, a také, jestli dané produkty jsou, nebo byly spárovány, či nikoliv. V reportu je také odkaz na excelovský soubor, ve kterém klient nalezne produkty, jež jsme se za daný měsíc pokusili spárovat, a zda toto párování bylo úspěšné.

5 Seznam zadaných úloh studentovi v průběhu odborné praxe

V této kapitole popíši úkoly, které jsem v rámci své praxe řešil. Úkolů bylo mnoho. Příkladem může být vytvoření REST API, jež umožňuje komunikaci backendu s ostatními částmi aplikace. Dalším příkladem je filtrování produktů v administraci. Toto bylo potřeba vyřešit například proto, aby bylo možno přednostně párovat produkty na základě konkrétního výrobce či kategorie. Za zmínku dále stojí spojení s nástrojem XeMeL, jenž řeší přegenerování feedů na základě již vytvořených pravidel. Jako poslední uvedu testy pomáhající předcházet tomu, že se v průběhu vývoje porouchá nějaká již fungující část aplikace. Úkolů bylo mnoho, jejich podrobný popis by přesahoval rámec této bakalářské práce, proto popíši jen ty nejobsáhlejší z nich.

5.1 Sběr dat

Abychom mohli párovat produkty, bylo nutné zajistit potřebná data z e-shopů a srovnávače. E-shopy posílají do srovnávače veškerá produktová data v XML feedu, data e-shopů jsme tedy získali jednoduše.

Dále bylo potřeba zjistit data srovnávače, tedy informace o produktových kartách, abychom věděli, kam produkt lze napárovat. I tento problém šlo vyřešit velice snadno. Zbožové srovnávače poskytují vlastní API. Používají je převážně bidovací nástroje k úpravě cen prokliků, případně nástroje k úpravě cen produktů na základě cen konkurence. Vzhledem k tomu, že společnost trayto a.s. má také nástroje sloužící k bidování, tak k tomuto API máme přístup.

Na toto API je možno zaslat požadavek, na který se vrátí odpověď ve formátu JSON. V odpovědi se však nevyskytují jen ceny nebo pořadí, jsou v něm i další vlastnosti, jako například název produktu, kategorie, výrobce a další. Toho jsme tedy využili a informace chtěli stáhnout odtud.

Problémem bylo, že dotazy na API lze provádět pouze na konkrétní produkt. Je tedy potřeba znát ID produktu ze srovnávače nebo URL jeho produktové karty, aby se později na danou produktovou kartu dalo dotazovat. Některé produkty se daly získat dotazem na kategorii, v jehož odpovědi bylo i top 50 produktů z dané kategorie. Toto by však bylo velmi málo. Proto jsme použili data z jiného našeho nástroje - BiddingTools. Ten se stará o automatický bidding a z jeho databáze jsme použili nemalý počet URL, ke kterým jsme následně stáhli data. Toto posloužilo jako základ pro naši databázi produktových karet.

Další produkty se nám podařilo získat z reportů, které srovnávače poskytují. Tyto reporty se generují alespoň 1x denně a jsou v nich informace o spárovaných produktech. Mezi těmito informacemi se nacházejí i ID produktů ze srovnávače či URL produktové karty.

Všechny tyto informace jsme tedy sloučili a postupně začali plnit naši databázi produktů. Po tom, co jsme data stáhli, zajistili jsme automatickou aktualizaci, abychom měli k dispozici aktuální informace.

K datům bylo potřeba zajistit ještě jeden poměrně důležitý prvek. Jedná se o obrázky. Podle obrázku dokážeme na první pohled poznat, jestli je produkt podobný, a poté můžeme

začít zkoumat, jestli je stejný, nebo zda se liší. Proto bylo logickým krokem obrázky umístit do administrace, aby sloužily jako pomoc pro uživatele, který bude produkty párovat.

Obrázky se dají snadno získat od e-shopu, ale také ze srovnávače. Jejich URL jsou obsaženy ve feedu i v odpovědi z API. Otázka tedy byla, jak obrázky budeme ukládat. Ukládání obrázků jako takové do databáze sice možné je, ale není to zrovna nejvhodnější ani nejpoužívanější přístup. Pro ukládání obrázků se obvykle používá poněkud jiný systém. Obrázek se uloží klasicky na disk, zatímco do databáze se ukládá pouze cesta k němu. V databázi tedy přibyde pouze cesta a k obrázku poté lze přistupovat velice jednoduše.

Obrázky jako takové však mohou zabírat mnoho místa na disku. Proto je bylo potřeba zmenšit. V administraci obrázek nezobrazíme v jeho původní velikosti, jelikož by to bylo značně nepřehledné. Pokud ho uživatel bude chtít vidět, může se prokliknout na produktovou kartu nebo do e-shopu a obrázky lépe porovnat. Proto před samostatným zápisem obrázků se nejdříve zmenší, a poté jsou uloženy na server.

Po sesbírání všech dat jsme potřebovali vyřešit jeden problém. Není potřeba snažit se o spárování všech produktů. Mnoho z nich již spárovaných je, nebo čeká na kontrolu, a tudíž by to pro ně bylo zbytečné. Proto jsme potřebovali odlišit produkty, pro které se mají hledat příležitosti ke spárovaní, a pro které ne. Pro všechny produkty jsme potřebovali zjistit status, jenž mají na srovnávači. Srovnávače poskytují přehledy, ve kterých jsou seznamy produktů podle statusů, jako například spárovaný produkt, nespárovaný produkt a podobně. Z těchto přehledů poté přiřadíme produktům status, na základě kterého se příležitosti hledají či ne.

Všechny tyto údaje jsme zařadili do pravidelné aktualizace, aby se párovaly pouze produkty, které jsou reálně ve feedu, ale také aby se nepárovaly již spárované produkty. Nyní jsme měli zajištěna všechna data a jejich pravidelnou aktualizaci. Mohli jsme tedy začít s pokusy o párování.

5.2 Párování na základě návrhů srovnávače

Srovnávač v administraci poskytuje soubor s reportem nespárovaných souborů. V tomto souboru jsou některé z produktů, jež mají status jiný než spárovaný. Jsou v něm základní informace o produktu, které jsou ve feedu. Je zde například ITEM_ID, jméno produktu, nebo URL produktu v e-shopu.

Kromě těchto základních informací jsou u některých produktů doplňkové informace, které mají pomoci s párováním produktů. Je zde uvedeno doporučené jméno, kategorie, EAN, a také URL produktové karty, na kterou by produkt měl být napárován. Tyto hodnoty jsou tu převážně pro produkty, které mají uvedeny EAN, ale jejich PRODUCTNAME a CATEGORYTEXT se liší od těch na srovnávači, a proto nebyly automaticky spárovány. To ale neznamená, že tyto údaje jsou špatně. Jde spíše o špatně pojmenované produkty, nebo o ty, které mají určitou vlastní kategorii. Tyto údaje jsou často správné, a pokud ne, bývá to způsobeno především nesprávným EAN kódem, jenž e-shop s daným produktem do srovnávače posílá.

Aktualizaci produktů jsme tedy rozšířili o zpracování tohoto souboru. Postupně projdeme všechny produkty, které se v souboru nacházejí. Pro ty z nich, které mají informace k párování, vyhledáme produktovou kartu v databázi, a pokud zde není, tak ji stáhneme. Následně tuto produktovou kartu nabídneme uživateli jako možnost k párování.

Tento soubor jsme použili jako výchozí bod pro naše návrhy. Jedná se o lehce dostupné informace, jež se pro párování dají využít. Navíc velká část z těchto návrhů je správně. Problémem je, že v tomto souboru je produktů poměrně málo. Navíc, pokud by nástroj bral data jen z tohoto souboru, nepřinášel by jako takový žádnou velkou přidanou hodnotu. Proto bylo potřeba hledat příležitosti na napárování v produktových kartách.

5.3 Párování na základě hledání v kartách

Po zpracování párování doporučených produktů bylo potřeba vymyslet algoritmus, který bude párovat všechny produkty a ne jen ty doporučené, jelikož v tomto je pravá hodnota nástroje. Nástroj by se měl pokusit vyhledat a nabídnout příležitost na napárování nejlépe u všech produktů, které nejsou spárované.

K tomuto účelu jsme shromažďovali informace o produktových kartách ze srovnávače. Hledání příležitostí opět probíhá pravidelně po provedení aktualizace produktů. Vyberou se produkty, které nejsou spárované, a s těmito se dále pracuje. Důležité také je, že příležitosti pro každý produkt se nehledají pokaždé. Pokud totiž nástroj nenalezne žádnou příležitost, je dost pravděpodobné, že se mu to nepodaří ani při dalším hledání. Proto se volí pouze ty produkty, pro které hledání již určitou dobu neproběhlo.

Na základě těchto produktů poté z databáze získáme produktové karty, na které by produkt mohl být napárován. Tato množina je poté procházena a porovnávána, a na základě těchto porovnání je vypočtena podobnost s těmito kartami. Porovnání probíhá na základě několika parametrů, jež jsou popsány v následujících podkapitolách. Porovnáním těchto parametrů získáme jejich vzájemné podobnosti. Součtem těchto podobností tedy získáme celkovou podobnost produktu a produktové karty.

5.3.1 Porovnání kategorií

Kategorie je první z parametrů, dle kterého se srovnávač pokouší produkt spárovat. [8] Je tedy logické, že na základě něj by se mělo párovat. Kategorie je navíc povinný element, takže bychom jej měli najít u každého produktu.

Produkt často nebývá spárován, jelikož nepoužívá kategorii podle stromu kategorií daného srovnávače, ale používá vlastní, stejnou jako v e-shopu. Tato kategorie se zásadně neliší. Například, pokud e-shop prodává kosmetiku, může prodávat krémy. E-shop k produktu pošle kategorii "Krása | kosmetika | krémy". Mohlo by se zdát, že kategorie je dostatečně vypovídající, ale například Heureka.cz rozlišuje, zda se jedná o krém pleťový, tělový nebo dětský. Může se tedy jednat o tři rozdílné kategorie.

Důležité je, že základ těchto kategorií je stejný. Podobnost se tedy vypočte hledáním společných slov v obou elementech CATEGORYTEXT - toho z e-shopu a toho ze srovnávače. Čím víc slov, či jejich částí je shodných, tím větší je jejich podobnost. Výsledný algoritmus můžeme vidět níže.

```
1  private function getWordSimilarityOfStrings(string $firstString, string $secondString) : int {
2    $firstString = $this->prepareString($firstString);
3    $secondString = $this->prepareString($secondString);
4    $wordsFromFirstString = explode(' ', $firstString);
5
6    $countOfSameWordsInStrings = 0;
7    foreach ($wordsFromFirstString as $word) {
8      $pattern = '/.*' . $word . '.*/';
9      if (preg_match($pattern, $secondString) != 0) {
10       $countOfSameWordsInStrings++;
11     }
12   }
13
14   return round($countOfSameWordsInStrings / count($wordsFromFirstString), 2) * 100;
15 }
16
```

Výpis 1: Funkce pro získání počtu stejných slov

Nejdříve jsou z řetězců odstraněny všechny znaky, které nejsou písmena nebo číslice. Dále jsou písmena obsahující háčky či čárky nahrazeny písmeny bez nich a jako poslední převedeme řetězce na malá písmena. Jeden z řetězců poté rozdělíme podle mezer, čímž získáme jednotlivá slova. Tato slova se následně pokoušíme hledat ve druhém řetězci. Nakonec vydělíme počet nalezených slov celkovým počtem slov. Toto číslo vynásobíme 100, čímž získáme podobnost v rozmezí 0-100.

5.3.2 Porovnání jmen

Dalším důležitým parametrem je nepochybně jméno produktu. Ze jména bychom totiž na první pohled měli poznat, o jaký produkt se jedná. Jedná se o element PRODUCTNAME z feedu. Pokud jej produkt nemá, bere se v potaz element PRODUCT. Jedná se o druhý z parametrů dle kterých srovnávač páruje. Opět je povinný, takže lze spoléhat na to, že bude ve feedu.

Porovnání jmen produktů do jisté míry funguje podobně jako funguje porovnání kategorií. Hledají se stejná slova, či jejich části, podle kterých se výsledná podobnost vypočte. Na rozdíl od kategorie však může být jméno produktu mnohem různorodější. Proto při porovnávání jsou použity další kroky, jež podobnost určí o něco přesněji.

Jedním z nich je hledání synonym a antonym. Při nalezení příležitostí dost často nastal případ, kdy byla nalezena a doporučena jiná varianta toho samého produktu. Například jiná velikost, barva a podobně. Tyto produkty jsou však ne vždy stejné. Některé z nich sice patří na stejnou produktovou kartu, ale jiné zase ne. Příkladem může být jiná velikost monitoru. Na

srovnávači jsou dva monitory od stejného výrobce, které se liší jen svou velikostí. Je logické, že oba monitory budou velmi podobné.

Za tímto účelem vznikl seznam synonym a antonym, který pomáhá vylučovat produkty, jež ve svém jménu mají antonymum. Tento seznam je pravidelně doplňován, aby tak byly omezeny příležitosti, které sice mají velice podobné parametry, ale jinak se jedná o dva odlišné produkty. Když se při porovnávání produktů objeví ve jménech antonyma, produktová karta není nabídnuta k párování.

5.3.3 Porovnání výrobců

Dalším parametrem, jenž je ve feedu posílán, je výrobce. Sice se nejedná o povinný element, ale do srovnávače je posílán téměř vždy, proto se jedná o další vhodný parametr na porovnání.

Porovnání výrobce bylo jedním z těch jednodušších. Jméno výrobce se nevymýšlí, ale je jasně dané. Proto se zde porovnává pouze to, zda jsou hodnoty stejné. Zároveň na jedné produktové kartě je vždy produkt od konkrétního výrobce. Na základě této hodnoty tedy lze vyřadit mnoho nevhodných příležitostí. Výslednou funkci můžeme vidět níže.

```
1  private function getSimilarityOfManufacturers(
2    string $feedManufacturer, string $aggregatorManufacturer
3  ) : int {
4    if ($feedManufacturer == null || $aggregatorManufacturer == null) {
5      return self::NO_MANUFACTURER_SIMILARITY;
6    }
7    $feedManufacturer = $this->prepareString($feedManufacturer);
8    $aggregatorManufacturer = $this->prepareString($aggregatorManufacturer);
9
10   return $feedManufacturer == $aggregatorManufacturer ? 100 : 0;
11 }
12
```

Výpis 2: Funkce pro porovnání výrobců

Funkce nejdříve kontroluje, jestli jsou dostupné hodnoty o výrobci pro produkt z e-shopu i pro produktovou kartu. Tato kontrola je nutná, jelikož výrobce není povinný parametr. Taktéž na některých produktových kartách není výrobce uveden. Pokud nastane tento případ, výrobce nelze porovnat, proto je použita základní podobnost. V případě, že jsou nastaveny obě hodnoty, jsou opět upraveny a zmenšeny. Poté výsledné řetězce porovnáme. Výsledná podobnost může být 0, nebo 100.

5.3.4 Porovnání cen

Cena, jakožto povinný element, bez kterého se produkt na srovnávač nikdy nedostane, je dalším elementem, dle něhož lze vyřadit mnoho nevhodných příležitostí. Ve feedu e-shopy posílají cenu, za kterou produkt nabízejí. Ta se pak propíše v produktové kartě.

Vzhledem k tomu, že každý e-shop si cenu určí sám, je zde určitý rozsah mezi minimální a maximální cenou. Pokud je tedy hodnota v uvedeném rozsahu, podobnost v rámci ceny je vysoká, nebo také maximální. Dále se podobnost postupně snižuje v závislosti na hodnotě, o kterou se cena liší.

```
1  private function getSimilarityOfPrices(
2    float $feedPrice, float $minPrice, float $maxPrice
3  ) : int {
4    if ($feedPrice >= $minPrice && $feedPrice <= $maxPrice) {
5      return 100;
6    } elseif ($feedPrice < $minPrice && $minPrice * self::LOW_PRICE_LIMIT < $feedPrice) {
7      return round(($feedPrice / $minPrice) * ($feedPrice / $minPrice), 2) * 100;
8    } elseif ($feedPrice > $maxPrice && $feedPrice < $maxPrice * self::HIGH_PRICE_LIMIT) {
9      return round(($maxPrice / $feedPrice) * ($maxPrice / $feedPrice), 2) * 100;
10   } else {
11     return 0;
12   }
13 }
14
```

Výpis 3: Funkce pro porovnání cen

Jako první porovnáváme, jestli je cena produktu z e-shopu mezi minimální a maximální cenou, za kterou produkt nabízí ostatní e-shopy na produktové kartě. Pokud ano, podobnost cen je maximální, tedy 100. Dále zkoušíme, jestli je cena mimo tento interval, ale pouze do určité míry. Toto se děje pro obě hranice intervalu. Pokud je některá z podmínek splněna, je vydělena menší hodnota tou větší. Toto číslo je vždy menší než 1. Pokud jej vynásobíme sebou samým, stane se menším než původně. Tímto zajistíme větší penalizaci pro hodnoty, které leží dále od hranic intervalu. Pokud cena produktu z feedu nespadá ani do jednoho z výše zmíněných intervalů, funkce vrací nulovou podobnost.

5.3.5 Porovnávání obrázků

Člověk za pomoci obrázků může velmi rychle rozhodnout, zda jsou dané produkty podobné či nikoliv. Vizuálně na první pohled rozhodne, že telefon a televize jsou dvě odlišné věci. Pro člověka se tedy jedná a poměrně triviální úkol. Naproti tomu pro počítač to již tak jednoduché není. Ale ačkoliv toto není jednoduché, není pravda, že by to bylo nemožné. Proto jsme se rozhodli, že produkty budeme porovnávat i na základě jejich obrázků. Obrázky jsme již k dispozici měli, nyní stačilo vymyslet jen metodu, jak se budou vzájemně porovnávat.

Nejdříve jsme na porovnání obrázků chtěli použít nějaký druh neuronové sítě, či jiný algoritmus založený na strojovém učení. Problém však byl, že s tímto ve firmě nikdo nemá žádné větší zkušenosti. Navíc by tento způsob byl poměrně časově náročný, a pravděpodobně ne tolik přínosný, vzhledem k vynaloženému úsilí. Museli jsme tedy najít jiný způsob.

Při hledání jsme našli, že pro podobnost obrázků se dá použít jejich hash. Toto mi však nejprve nedávalo smysl. S hashováním se nejčastěji setkáváme u hesel, nebo obecně informací, které chceme určitým způsobem skrýt. Použitím hashovací funkce na jeden řetězec bychom měli pokaždé dostat ten samý hash. Pokud se ale bude řetězec lišit v jediném znaku, měl by jeho hash být naprosto jiný. Z toho vyplývá, že tento způsob by se dal použít, pokud bychom měli stejný obrázek, ovšem pokud by byl obrázek už jen trochu jiný, bude úplně jiný i jeho hash. [11]

Pro hashování obrázků se však používají jiné, méně známé hashe, které fungují na jiných principech než ty, jež mají za účel hashovat například zmíněná hesla. Hashe dvou podobných obrázků si budou podobné také. Tedy lze spočítat i jejich podobnost. Výsledkem těchto hashů bývají velmi často řetězce, jež obsahují pouze 0 a 1. To proto, aby se dané řetězce daly snadno porovnat. [11]

Nyní již stačilo zjistit, jak se podobnost dvou hashů dá spočítat. Toto byla nejjednodušší část na celém procesu hashování obrázků. Hashe tvořené nulami a jedničkami se dají porovnat za pomoci Hammingovy vzdálenosti, která funguje na jednoduchém principu.

5.3.5.1 Hammingova vzdálenost

Hammingova vzdálenost se používá pro vypočtení rozdílu dvou řetězců. Výsledkem je počet znaků, které je potřeba v jednom či druhém řetězci změnit, aby byly shodné. Tuto hodnotu lze vypočítat pro jakékoliv dva řetězce. Jedinou podmínkou je, že jejich délka musí být shodná. [12]

Můžeme si to ukázat na následujícím příkladu. Vezměme si dvě náhodná, ale vizuálně podobná binární čísla, resp. řetězce.

110110100101
110010110101

Řetězce jsou stejně dlouhé, takže můžeme vypočíst jejich Hammingovu vzdálenost. Pokud se na tyto dva řetězce podíváme, zjistíme, že jsou si velice podobné. Nyní budeme jeden z řetězců procházet znak po znaku, a ten pak porovnávat se znakem, který je na stejné pozici v řetězci druhém. Pokud jsou znaky stejné, přejdeme k dalšímu znaku, ale pokud jsou rozdílné, zvyšuje se jejich vzdálenost o hodnotu 1. Poté, co projdeme celý řetězec, zjistíme, že Hammingova vzdálenost těchto dvou řetězců je rovna 2. A to právě proto, že se liší znaky na 3. a na 7. pozici.

110110100101
110010110101

Změníme-li označené znaky, řetězce budou totožné.

110110100101
110110100101

Tímto způsobem tedy můžeme porovnat dva jakékoliv obrázky, které jsou zahashované stejným hashovacím algoritmem. Teď již jen zbývalo zjistit, jaký hashovací algoritmus bude pro náš případ vhodný. Začali jsme tedy hledat, jaké algoritmy se dají pro hashování použít.

5.3.5.2 Average hash

První hash, který jsme se rozhodli implementovat, se nazývá Average hash. Tento hash je vytvořen z průměrné barvy obrázku. Tato hodnota je poté porovnávána s jednotlivými částmi (výřezy) a podle toho se určuje hash daného obrázku.

Na srovnávačích mají všechny obrázky jednotnou podobu. Uprostřed je produkt a kolem něj se nachází bílé pozadí. [13] Je tedy jisté, že produkt bude uprostřed a kolem nebude nic jiného. Pokud tedy vypočítáme průměrnou barvu obrázku, tak je jasné, že ta bude silně ovlivněna i pozadím. To se na první pohled může zdát jako nevýhoda, ale tato skutečnost se dá využít.

Princip hashe je velmi jednoduchý. Porovnáváme barvu konkrétního pixelu s průměrnou barvou celého obrázku. Na základě toho poté dostaneme výsledný hash. Je-li barva daného pixelu tmavší než barva obrázku, přidáme k hashi 1. V opačném případě přidáme 0. Takto projdeme celý obrázek a dostaneme výsledný hash.

Tento postup není nijak složitý, nicméně narazíme zde na určitý problém. Porovnávat všechny pixely by bylo časově velmi neefektivní, výsledný hash by byl příliš dlouhý a pravděpodobně ani výsledek by nebyl tak kvalitní. Navíc bychom mohli porovnávat pouze obrázky, které jsou stejně velké. Tím bychom nemohli porovnat všechny obrázky, ale jen stejně velké skupiny. Dalším problémem je, jak porovnat například žlutou barvu se zelenou. Celý algoritmus se tedy skládá z více kroků než ze samotného porovnávání barev a následného získání hashe.

Předveďmě si celý proces na obrázku. Máme obrázek, jehož velikost je 130x130 pixelů. Toto je velikost zmenšeného obrázku ze srovnávače. Dále zmíním, že přiložený obrázek je rozostřený. To je způsobeno tím, že je oproti původnímu obrázku zvětšený na velikost 260x260 px, aby byl lépe čitelný.

Obrázek 1: Obrázek, který budeme hashovat

V prvním kroku se obrázek převede na odstíny šedi. Toto se provede kvůli tomu, že porovnávat odstíny šedi je jednodušší, než porovnávat modrou barvu s červenou a podobně. Výsledkem je stejně velký obrázek v černobílé podobě.

Tímto krokem se vyřeší problém s porovnáním různých barev. Porovnání stupňů šedi je jednodušší proces než porovnávání barev ve formátu RGB. Porovnání obrázku se nyní značně zjednoduší. Vypočteme průměrnou hodnotu stupně šedi pro celý obrázek a poté budeme hodnoty vzájemně porovnávat.

Problém, který zatím zůstává, je velikost obrázku. Počet pixelů, které tvoří tento obrázek, je 16900 (130x130). Při dnešním výpočetním výkonu by porovnání dvou řetězců o takové velikosti

časově tolik náročné nebylo. Ale pokud bychom měli porovnat takových obrázků několik tisíc, již by tato operace mohla určitou dobu trvat. Nehledě na to, že ukládat řetězce o takové délce do databáze není zrovna efektivní. Navíc pořád lze porovnat jen obrázky stejné velikosti.

V následujícím kroku se tedy obrázky převedou na jednotnou velikost. Důležitým krokem je i to, aby obrázky, které nemají poměr šířky a výšky 1:1, byly před samostatným zmenšením doplněny o bílé pozadí. Tím docílíme toho, že při zmenšení se obrázky nedeformují. V tomto kroku obrázky zmenšíme, čímž dostaneme jednotnou velikost všech obrázků, a tudíž splníme podmínku pro výpočet Hammingovy vzdálenosti. Důležité je také podotknout, že tímto zmenšením velikostí velice zredukujeme informace obsažené v obrázku, a to do té míry, že nepůjde poznat, co na obrázku bylo původně. Výsledek bude vypadat takto.

Obrázek 2: Obrázek, zmenšený na velikost 16x16px

Získali jsme obrázek, jehož velikost je 16x16 pixelů. Počet pixelů tohoto obrázku je 256. Obrázek v současné podobě je již nečitelný. Těžko bychom poznali, co na něm bylo původně. Ale dostali jsme se na velikost, se kterou se dá pracovat daleko lépe než s původní. Další důležitý fakt je ten, že zmizely i některé odlišnosti v detailech, čímž se usnadní porovnávání. Obrázek nyní lze zahashovat. Hash vypadá takto:

1111111000111111111111101011111111111110101111111111110010111111111111110111111
1111111101111111111111110111111111111010111111111111100111111111111000111111111111
1100000011111111110000000101111110000000010111110000000000011110000111000001111111
1100110011

Pokud si hash převedeme do čtverce, lze v něm částečně vidět původní tvar.

1111111000111111
1111111010111111
1111111010111111
1111110010111111
1111111110111111
1111111110111111
1111111110111111
1111110101111111
1111111001111111
1111100011111111
1111100000011111
1111100000001011
1111000000001011
1110000000000011
1100001110000011
1111111100110011

Hash je nyní kompletní. Takto se v praxi hashují obrázky, aby se poté jednoduše daly porovnat. Nicméně v našem konkrétním případě jsme museli provést jednu drobnou úpravu. Z každé strany jsme ubrali 2 pixely. Výsledný hash tedy vypadá takto:

111110101111
111100101111
111111101111
111111101111
111111101111
111101011111
111110011111
111000111111
111000000111
111000000010
110000000010
100000000000

Tuto úpravu jsme provedli kvůli tomu, abychom se zbavili volného prostoru kolem. Ztratili jsme sice informace z hashe, nicméně v našem případě začala podobnost vycházet o něco lépe. Naším výsledkem je tedy hash 144 znaků dlouhý.

Dále dodám, že velikost i ořezání hashe jsme upravili podle naší potřeby. Typický hashovací algoritmus převede obrázek na velikost 8x8 pixelů, tudíž délka výsledného hashe je 64 znaků. Po určité době testování jsme zjistili, že pro nás je to málo, a proto jsme se rozhodli velikost zvětšit. Taktéž ořezání není základní součástí algoritmu, ale pro tyto konkrétní obrázky byly výsledky porovnávání o něco lepší.

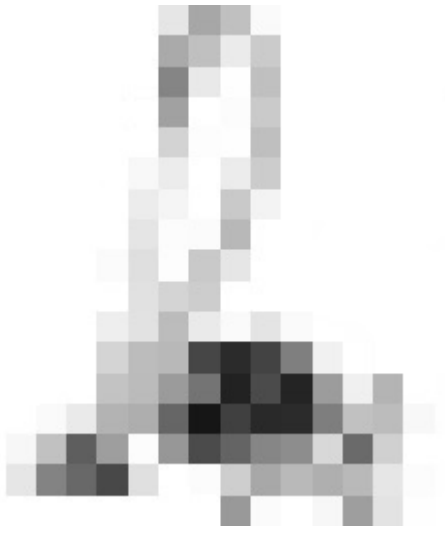
To je tedy první hash, který jsme pro porovnání použili. Takto použitý hash dokáže poměrně přesně porovnat tvar na obrázku. Dokáže tedy rozeznat, jestli máme na obrázku tričko nebo kalhoty a podobně. Nicméně již nepozná, jestli je uprostřed trička potisk, nebo jestli je čisté.

5.3.5.3 Difference hash

Kvůli tomu jsme se rozhodli, že pro porovnání použijeme ještě jeden hash, který toto dokáže alespoň trochu zohlednit. Použili jsme tedy druhý hash. Jedná se o Difference hash. Tento hash opět porovnává barvu pixelů. Na rozdíl od Average hashe, neporovnává barvu pixelu s průměrnou barvou celého obrázku, ale s předchozím pixelem ve stejném řádku. Pokud je pixel tmavší, přidá k hashi 1, v opačném případě přidá 0.

Základ tohoto hashe je tedy stejný. Nejdříve převede obrázek do odstínů šedi. Poté obrázek zmenší, a na základě tohoto zmenšeného obrázku vypočte hash, který pro naše účely opět ořežeme. Všechny kroky jsou tedy stejné. Liší se velikost, na kterou obrázek zmenší a také v konstrukci samotného hashe.

Po odebrání barev tedy obrázek opět zmenšíme. Nicméně jej nezmenšíme na velikost 16x16 pixelů, ale do velikosti 17x16 pixelů. Šířka tedy bude o 1 pixel větší. Tento sloupec pixelů navíc je zde proto, abychom dodrželi výslednou velikost hashe, tj. 144 znaků. Výsledný obrázek vypadá takto:



Obrázek 3: Obrázek, zmenšený na velikost 17x16px

Opět z obrázku již nepoznáme, co na něm bylo původně. Ale snáze takto zkonstruujeme hash. Začneme tedy s pixelem v 0. řádku na 1. pozici. Porovnáme jej s pixelem, který je na 0. řádku na 0. pozici. Pokud je tmavší, připíšeme k hashi 1, v opačném případě 0. Jako další porovnáme hodnotu na 2. pozici s hodnotou na 1. pozici a opět přidáme k hashi 0 nebo 1. Takto to provedeme pro celý řádek. Následně toto samé provedeme pro všechny zbylé řádky. Dostaneme tedy opět hash, jenž bude dlouhý 256 znaků. Bude vypadat takto:

0000001100000000
0000001001000000
0000001001000000
0000001001000000
0000001011000010
0000011011000000
0000010010000000
0000010110000000
0000010100010010
0000111000000000
0000111101000000
0000111110100100
0001101100100100
0111001100001000
0111100111010000
0100000010011000

Po jeho ořezání dostaneme tento hash.

000010010000
000010010000
000010110000
000110110000
000100100000
000101100000
000101000100
001110000000
001111010000
001111101001
011011001001
110011000010

Opět jsme dostali hash 144 znaků dlouhý. Hash sice není již tolik efektivní jako Average hash a slouží spíše jako jeho doplněk, ale i tak je část podobnosti je určena i na jeoh základě.

Poté, co jsou sesbírány všechny podobnosti, jsou tyto podobnosti vynásobeny váhou, která určuje, jaký vliv má daná podobnost na celkovou podobnost produktu. Tyto vynásobené podobnosti sečteme, čímž získáme celkovou podobnost produktu a produktové karty.

```
public function getSimilarity(
  FeedProduct $feedProduct, AggregatorProduct $aggregatorProduct,
  array $aggregatorProductImages
) : int {
  $feedProductName = $feedProduct->getProductName() == null ?
    $feedProduct->getProduct() : $feedProduct->getProductName();
  $productNameSimilarity = $this->getSimilarityOfProductNames(
    $aggregatorProduct->getProductName(), $feedProductName);
  $manufacturerSimilarity = $this->getSimilarityOfManufacturers(
    $aggregatorProduct->getManufacturer(), $feedProduct->getManufacturer());
  $aggregatorCategoryFullName = $this->aggregatorCategories[
    $aggregatorProduct->getCategoryUuid()]->getCategoyFullName();
  $categorySimilarity = $this->getSimilarityOfCategories($aggregatorCategoryFullName,
    $feedProduct->getCategoryText());
  $priceSimilarity = $this->getSimilarityOfPrices($feedProduct->getPriceVat(),
    $aggregatorProduct->getMinPrice(), $aggregatorProduct->getMaxPrice());
  $imageSimilarity = $this->getImageSimilarityOfImages($aggregatorProductImages,
    $feedProduct->getImageAverageHashColor(), $feedProduct->getImageAverageHash(),
  $feedProduct->getImageDifferenceHash());

  return round(
      $imageSimilarity * self::IMAGE_SIMILARITY_WEIGHT
    + $productNameSimilarity * self::PRODUCT_NAME_SIMILARITY_WEIGHT
    + $manufacturerSimilarity * self::MANUFACTURER_SIMILARITY_WEIGHT
    + $categorySimilarity * self::CATEGORY_SIMILARITY_WEIGHT
    + $priceSimilarity * self::PRICE_SIMILARITY_WEIGHT
  );
}

```

Výpis 4: Funkce pro získání podobnosti produktu a produktové karty

Po prozkoumání všech nabízených produktových karet se použijí dvě nejlepší z nich, a pokud přesahují určitou hraniční hodnotu, tak se uloží jako příležitosti ke spárování. Tyto příležitosti poté uživatel projde a rozhodne o jejich správnosti.

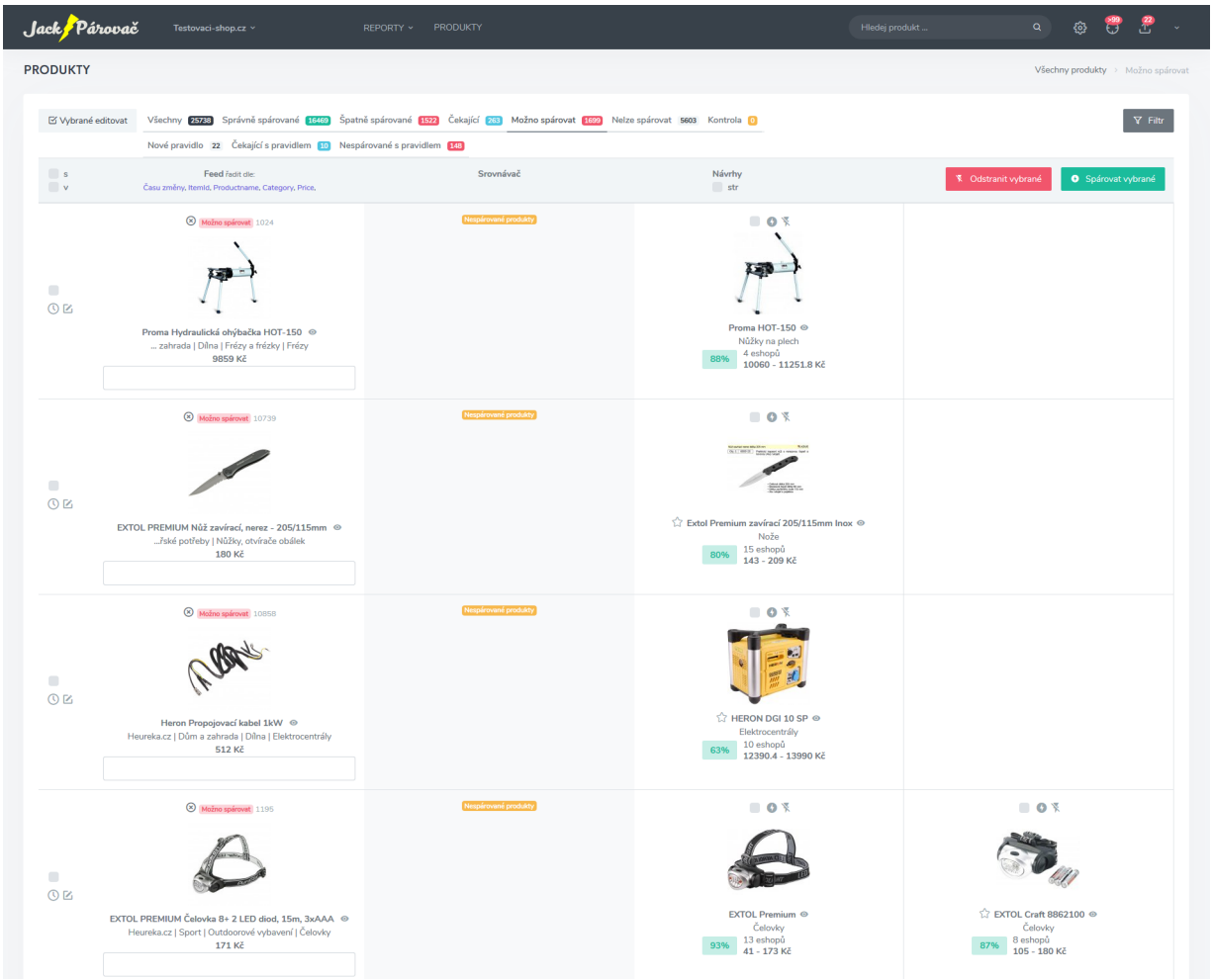
5.4 Uživatelské rozhraní

Když byl systém pro hledání příležitostí hotov, bylo potřeba vytvořit rozhraní, díky kterému může uživatel nástroj používat. Bylo tedy zapotřebí vyvinout frontendovou část aplikace.

Vzhledem k tomu, že nástroj bude minimálně v nejbližší době fungovat pouze interně, není potřeba, aby systém zaujal jedinečným designem, důležitá je především jeho funkčnost. Proto v rámci časové úspory byla zakoupena šablona, která poskytovala příjemný design, ale také poměrně mnoho již hotových řešení. Použili jsme ji jako základ pro naše rozhraní. Vzali jsme části, které se nám hodily, a ty jsme pak doplnili o naše specifické potřeby.

Toto rozhraní jsme poté napojili na backendové API, doplnili jej o JavaScriptové prvky a základní funkce byly hotovy. Vzniklo tedy jednoduché rozhraní splňující všechny naše požadavky.

K tomuto rozhraní bylo potřeba doplnit další funkcionality do backendové části aplikace. Šlo například o rozhodnutí o správnosti návrhu a následnému vytvoření pravidla, rozhodnutí o nesprávnosti a následnému zamezení opakovaného návrhu stejné karty, filtrování produktů a podobně.



Obrázek 4: Ukázka rozhraní

Výsledkem je toto rozhraní. V této části aplikace pracují uživatelé nejvíce. Jedná se o stránku,

kde se párují produkty. Nahoře je menu, kde můžeme přepínat mezi jednotlivými e-shopy, přejít na stránku s produkty či reporty a podobně.

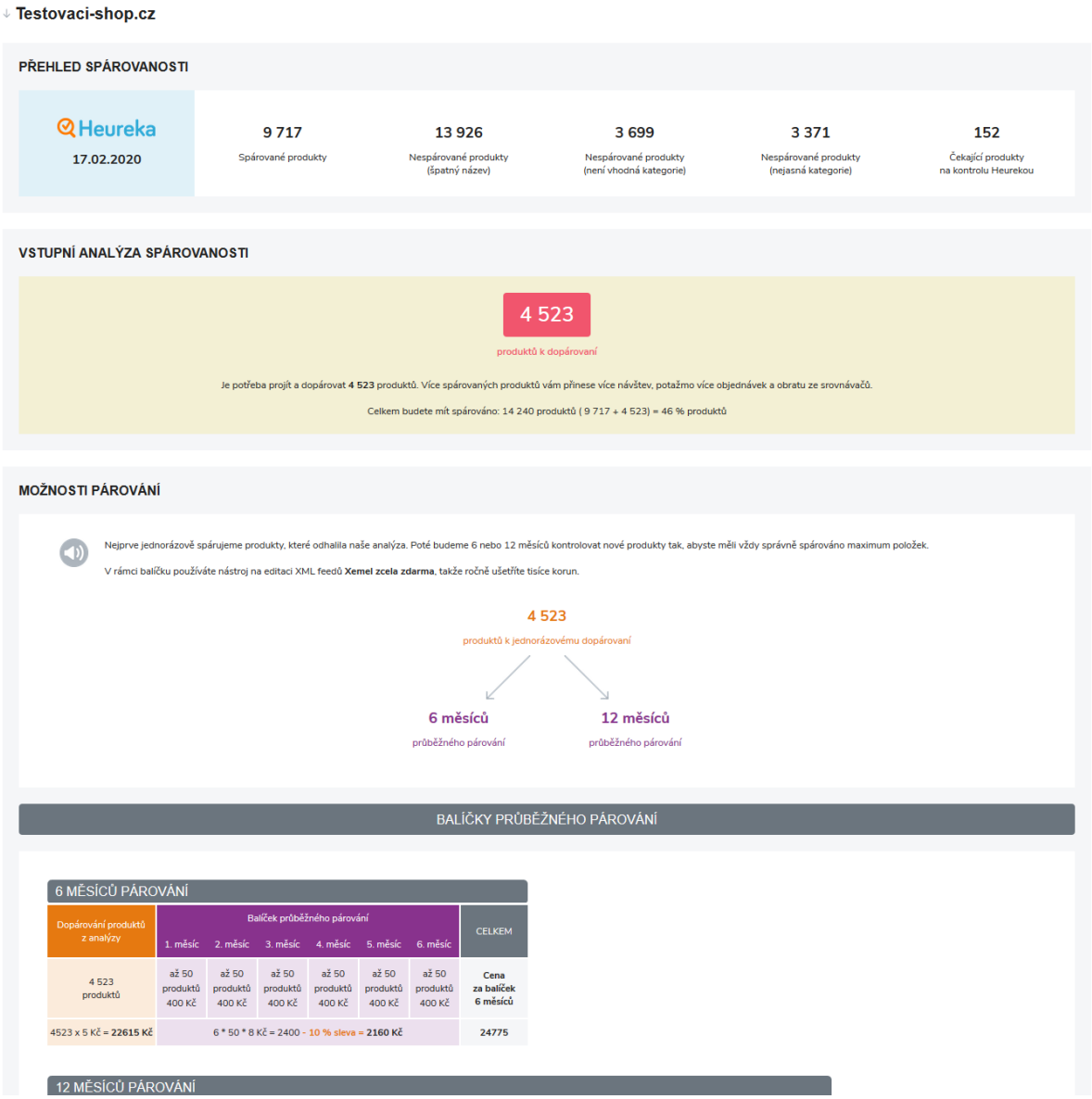
Na stránce můžeme vidět sekci s produkty. Nahoře lze vybrat produkty dle jejich statusů, které jim nástroj přiřadí. Napravo lze otevřít filtr, kde produkty můžeme následně filtrovat dle jejich ID, jména, ceny a podobně. Pod tímto řádkem se nachází produkty. Tyto produkty jsou uspořádány v tabulce. Každý řádek tabulky je 1 produkt, který e-shop posílá ve feedu do srovnávače. Tabulka jako taková se skládá z 5 sloupců. V prvním sloupci jsou umístěny pouze ikony, dle nichž lze zobrazit historii produktu, nebo jej lze přímo upravit. Ve druhém sloupci jsou produkty tak, jak je e-shop posílá do srovnávače. Jsou zde základní informace jako jméno, kategorie nebo cena. Zbylé 3 sloupce jsou téměř stejné. Jsou v nich produkty ze srovnávače. Nachází se v nich jméno, kategorie, počet e-shopů na produktové kartě a jejich cenový rozsah. V prvním z nich (tedy 3. v tabulce) je produktová karta, na kterou je produkt napárován. Tento sloupec slouží převážně pro produkty, které jsou již spárovány. V posledních dvou sloupcích jsou příležitosti, kam můžeme produkt napárovat. Jsou zde maximálně dvě příležitosti k napárování. Nad obrázkem jsou poté dvě ikony, z nichž jedna slouží pro spárování a druhá pro označení špatného návrhu.

5.5 Analýza

Další nemalou součástí, kterou bylo potřeba vyřešit, byly analýzy. Analýza má za úkol potenciálnímu klientovi ukázat, jaké výsledky mu může nástroj přinést. Na základě ní se klient rozhodne, zda služby využije či ne. Bylo tedy potřeba vymyslet postup, který zjistí počet produktů, jenž jsme schopni spárovat. Tento systém měl být přesný. Pokud bychom totiž podcenili počet, který dokážeme spárovat, potencionálního klienta by to nemuselo přesvědčit. V opačném případě, kdyby naše čísla byly příliš vysoká, tak by klient mohl očekávat něco, co bychom mu za pomoci nástroje nebyli schopni dodat.

Bylo tedy nutné vymyslet, jak potřebná čísla získat. Počet produktů, jež se dá spárovat, se ale dá získat velice snadno. A to natažením produktů a pokusem o hledání příležitostí. Ve chvíli, kdy se tedy klient rozhodne o zaslání analýzy, zjistíme, kolik příležitostí k jeho produktům jsme schopni nalézt. Na základě toho vypočteme, kolik návrhů by mohlo být správných. Tato čísla poté zobrazíme v analýze, a tu zašleme klientovi.

Analýza však není jen zaslání několika čísel. Bylo tedy potřeba ji zobrazit. Některé údaje v analýze se negenerují, ale jsou vyplňovány ručně obchodním oddělením. Proto pro tyto potřeby bylo nutné zajistit i editaci některých údajů v analýze. Vytvořili jsme tedy jednoduchou jednostránkovou aplikaci. Ta má dva úkoly. Prvním z nich je získání dat z backandového API na základě hashe, který má v URL. Tato data, jež přijdou jako odpověď, poté zobrazuje. Druhým úkolem je zajištění editace některých hodnot v analýze, jako je cena. Tato úprava je možná při připojení z vnitřní sítě a je dostupná v administraci nástroje.



Obrázek 5: Analýza

Analýza vypadá takto. Nahoře se nachází počty produktů dle toho, jaký mají status na srovnávači. Pod ním se nachází část, kde je napsáno, kolik je nástroj schopen dopárovat produktů. Jedná se tedy o přibližný počet příležitostí, který nástroj našel. Klient zde uvidí, jakou výslednou spárovanost může mít, a kolik produktů bude spárováno. Pod tímto se nachází možnosti párovacích balíčků k zakoupení, jejich kompletní přehled a cena.

5.6 Měsíční report

Další poměrně rozsáhlou funkcionalitou je měsíční report. Ten slouží jako zpětná vazba pro klienta. Může z něj vyčíst, kolik produktů jsme se pokusili spárovat, vývoj počtu jeho produktů v XML feedu v čase a podobně. K tomu, abychom tato čísla získali, bylo potřeba si ukládat, co se s jednotlivými produkty v čase děje. Museli jsme proto vytvořit systém logování, ve kterém budou uloženy všechny důležité změny, a na základě nich poté tyto hodnoty počítat.

Nejdřív jsme se museli rozhodnout, které údaje pro vytvoření reportu budeme potřebovat. Dále jsme se rozhodli, že tento log nebudeme využívat pouze pro report. Chtěli jsme zaznamenat i něco navíc, dle čeho bychom mohli zpětně zjišťovat, proč byly navrženy některé návrhy, jak dlouho trvalo, než se produkt spároval a podobně. Výhodou bylo, že všechny tyto události již určitým způsobem probíhaly, takže nebylo potřeba zajišťovat žádná nová data, ale pouze systém, který se bude využívat pro jejich ukládání.

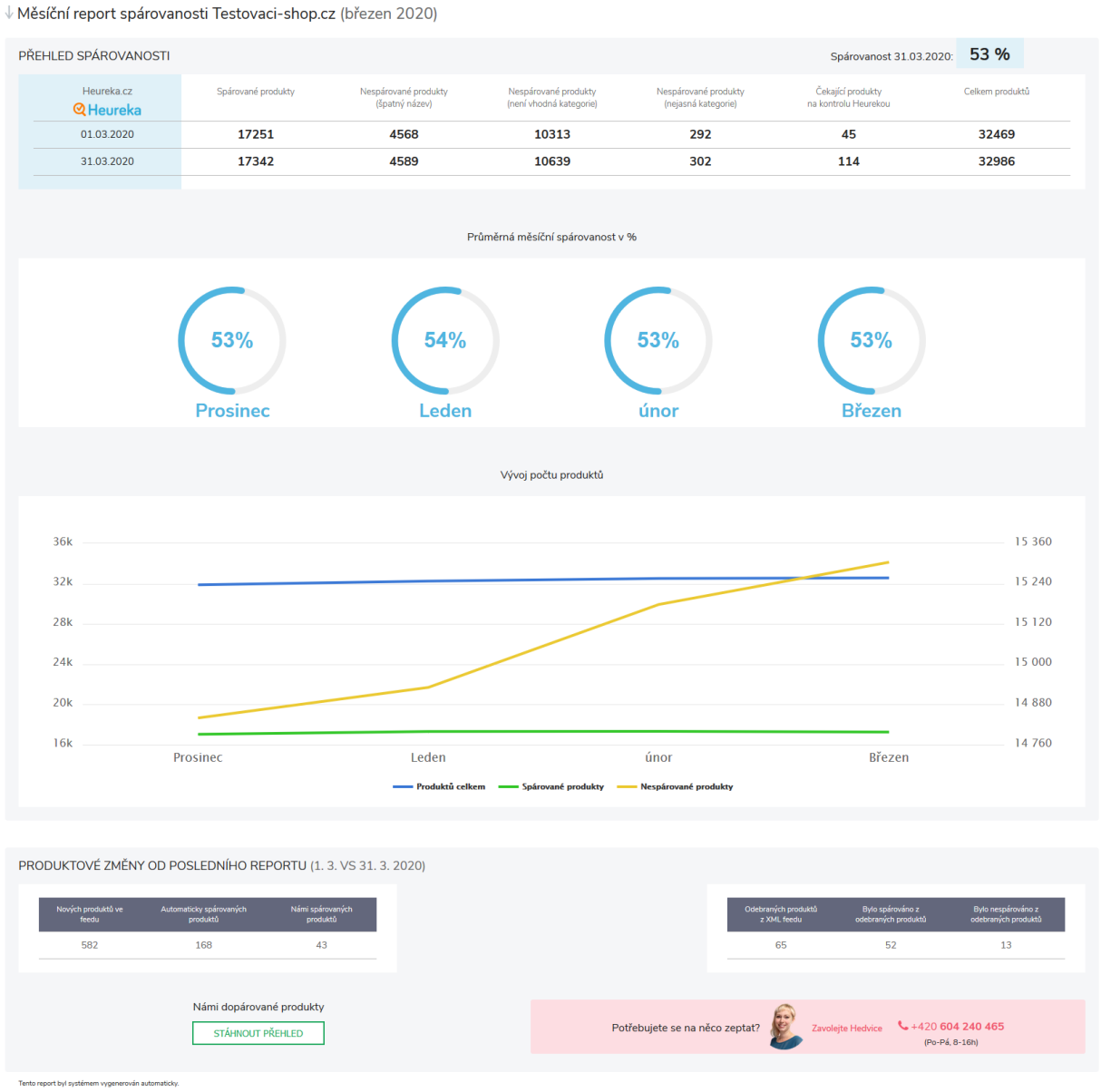
V těchto datech jsme potřebovali hledat, proto bylo logickou volbou ukládání do databáze. Vzhledem k tomu, že všechny události jsou spojeny s ukládáním či vkládáním dat do databáze, skvěle se pro tuto operaci osvědčilo použití databázových triggerů, v některých případech pak volání procedur. Odpadla tak povinnost v kódu porovnávat 2 produkty - jeden před změnou a druhý po ní - a následné rozhodování o potřebě uložení této události. V triggeru po updatu jsou obě tyto informace k dispozici, proto je tato operace na tomto místě mnohem snazší.

Následně se napsaly triggery a procedury, a vznikl tak systém logování, ve kterém byly všechna pro nás důležitá data o historii produktu. Poté byla tato funkcionalita přidána do administrace a zbývalo vytvořit zmíněný report.

Report se do jisté míry podobá analýze. Jedná se o jednostránkovou aplikaci, která má sloužit ke zobrazení dat. Na rozdíl od analýzy se však jedná o report historie, proto zde není potřeba cokoliv měnit. Šlo pouze o připojení se na API, získání dat a jejich následné zobrazení.

V čem se však report výrazně liší od analýzy, je jeho vytváření. Narozdíl od analýzy, kde bylo potřeba dokončit pouze pár jednoduchých výpočtů, bylo vytvoření reportu složitější. Při jeho vytvoření se nejprve musí vybrat z databáze všechna potřebná data z logu za poslední měsíc. Tyto logy jsou poté seskupeny po dnech. Dny se poté prochází a počítá se, kolik bylo ve který den ve feedu produktů, kolik jich přibylo, nebo kolik zmizelo. Zároveň se zjišťuje, kolika produktům bylo daný měsíc vytvořeno či jiným způsobem upraveno pravidlo. Z těchto čísel jsou poté vypočteny hodnoty počtů produktů, jejich spárovanosti, či jejich výsledný pohyb za měsíc.

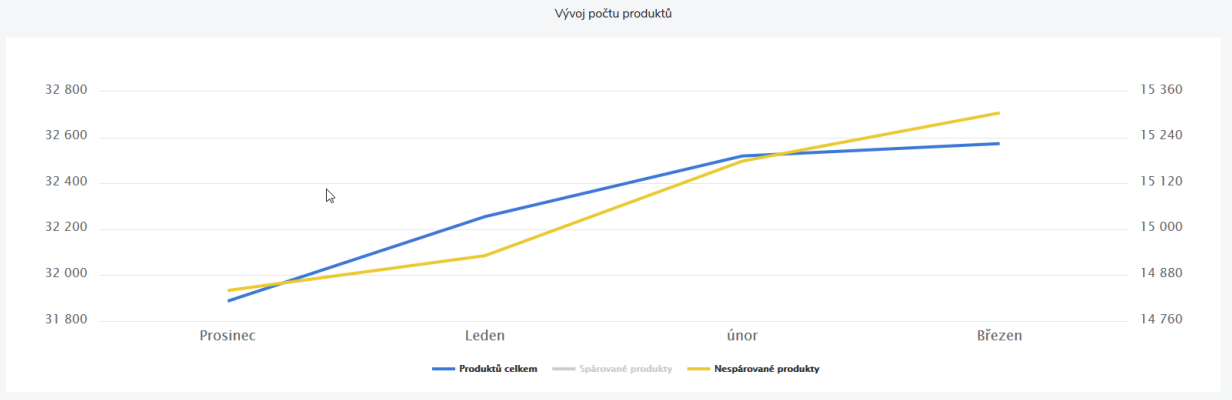
K reportu se navíc generuje excelovský soubor, ve kterém jsou vypsány všechny produkty, k nimž bylo v uplynulém měsíci vytvořeno pravidlo. Tento soubor poté slouží jako zpětná vazba, kolik produktů jsme v uplynulém měsíci spárovali. Soubor se dále ukládá na serveru a k jednotlivým reportům je uložena pouze cesta k nim, obdobně, jako jsou uloženy obrázky.



Obrázek 6: Měsíční report

Na začátku reportu je opět počet produktů dle jejich statusů na srovnávači. Narozdíl od analýzy jsou zde dvě hodnoty, aby klient mohl porovnat začátek a konec měsíce. Pod těmito údaji se nachází průměrná spárovanost v posledních měsících. Vzhledem k tomu, že procentuální hodnota spárovanosti se tolik nehýbe, je zobrazena pouze procentuálně. Pod tímto zobrazením se nachází graf. Graf zachycuje počet produktů v čase.

Pro e-shopy s velkým objemem produktů je křivka často celkem přímá. Proto je zde možnost křivky vypínat či zapínat, čímž se graf překreslí. Takto z něj lze lépe číst. Graf po vypnutí křivky Spárované produkty vypadá takto.



Obrázek 7: Graf vývoje počtu produktů

Pod grafem se nachází počty přidaných či odebraných produktů, kolik jich spároval srovnávač, nebo kolik jich bylo spárováno za pomoci nástroje. Nachází se zde také možnost stáhnout soubor, ve kterém je přehled námi spárovaných produktů.

6 Uplatněné a chybějící znalosti

Vzhledem k tomu, že ve společnosti již určitou dobu pracuji, mnoho znalostí jsem zde již pochytil. Příkladem může být například prohloubení znalostí ohledně jazyka PHP. Ve firmě jsem se také naučil používat verzovací systém Git, který jsem do té doby neznal a během studia jsem se s ním příliš nesetkal, což je dle mého názoru škoda.

6.1 Uplatněné znalosti

Mezi uplatněné znalosti, které jsem se naučil v době studia, patří záležitosti týkající se návrhu softwaru, které byly popsány v předmětu Vývoj informačních systémů. I když byly tyto znalosti probírány spíše teoreticky, využil jsem některé z probíraných návrhových vzorů, které mi pomohly psát čitelnější a rozšiřitelnější kód.

Dalším příkladem může být použití databázových procedur a triggerů, které jsem se naučil v předmětu Databázové a informační systémy. Tyto nástroje jsem předtím neznal a jejich použití mi ulehčilo mnoho práce.

6.2 Chybějící znalosti

V průběhu praxe mi nejvíce chyběly znalosti ohledně testování softwaru. S tímto jsem se setkal pouze v předmětu Architektura technologie .NET, kde byly zmíněny unit testy. Unit testy jsou ovšem jen některé z možností pro testování softwaru. Myslím si, že toto téma by mohlo být vyučováno alespoň v některém z volitelných předmětů. Díky psaní testů jsem také lépe pochopil, proč by se kód měl dělit, proč je vhodné použít některé návrhové vzory a podobně. Proto se domnívám, že by se mohlo jednat o přínosný předmět.

7 Závěr

Cílem této práce, bylo popsat mou pracovní náplň během bakalářské praxe. Hlavním projektem, a tedy i výsledkem, je funkční software, jenž pomáhá s párováním produktů na zbožových srovnávačích. Ačkoli ve vývoji softwaru budu i nadále pokračovat, nástroj již funguje komerčně a prodává se jako služba klientům. Do budoucna se bude pravděpodobně rozšiřovat o nové funkcionality, nebo budou ty stávající rozšířeny, či vylepšeny. Taktéž bude dokončena možnost párování produktů na Zboží.cz.

Během praxe jsem si prohloubil své znalosti v programovacích jazycích PHP a JavaScript. Taktéž jsem si zdokonalil své dovednosti ohledně databází, či práce s operačním systémem Linux. Lépe jsem pochopil, jak správně navrhovat, vyvíjet a testovat software. Poznal jsem také některé z nevýhod agilního systému řízení projektů, kdy jsem některé části systému více či méně musel přepsat kvůli změně požadavků.

Celkově svou praxi ve firmě hodnotím kladně. Získal jsem mnoho nových dovedností a prohloubil, či vyzkoušel jsem si ty stávající. Zjistil jsem, jak probíhá vývoj softwaru, a také to, co vše se děje kolem tohoto procesu. Po odpracování dnů v rámci této praxe jsem zůstal ve firmě a pokračuji ve vývoji tohoto i dalších nástrojů.

Literatura

1. *biddingtools.cz* [online]. trayto a.s. [cit. 2020-04-13]. Dostupné z: https://biddingtools.cz/.
2. *ettecapital.com* [online]. ette capital a.s. [cit. 2020-04-13]. Dostupné z: https://www.ettecapital.com/.
3. MYSLÍN, J. *Scrum*. Computer Press, 2017. ISBN 9788025146552.
4. *Slim framework* [online]. Slim Framework Team [cit. 2020-04-13]. Dostupné z: http://www.slimframework.com/.
5. *Codeception* [online]. Codeception Team [cit. 2020-04-13]. Dostupné z: https://codeception.com/.
6. *Heureka.cz* [online]. Heureka Group a.s. [cit. 2020-04-13]. Dostupné z: https://www.heureka.cz/.
7. *blog Heureka.cz EAN kódy na Heurece* [online]. Heureka Group a.s. [cit. 2020-04-13]. Dostupné z: https://blog.heureka.cz/2019/02/28/lekce-8-ean-kody-na-heurece/.
8. *blog Heureka.cz Jak funguje párovací proces na Heurece* [online]. Heureka Group a.s. [cit. 2020-04-13]. Dostupné z: https://blog.heureka.cz/2018/10/23/jak-funguje-parovaci-proces-na-heurece/.
9. *Heureka.cz - Specifikace XML feedu* [online]. Heureka Group a.s. [cit. 2020-04-13]. Dostupné z: https://sluzby.heureka.cz/napoveda/xml-feed/.
10. *blog Heureka.cz - PRODUCTNAME vs. PRODUCT* [online]. Heureka Group a.s. [cit. 2020-04-13]. Dostupné z: https://blog.heureka.cz/2018/11/20/lekce-3-productname-vs-product/.
11. *Different image hash functions* [online]. The Content Blockchain Project [cit. 2020-04-13]. Dostupné z: https://content-blockchain.org/research/testing-different-image-hash-functions/.
12. SURHONE, L.M.; TENNOE, M.T.; HENSSONOW, S.F. *Hamming Distance*. Betascript Publishing, 2010. ISBN 9786133001770.
13. *blog Heureka.cz - Jak na obrázky a popisky* [online]. Heureka Group a.s. [cit. 2020-04-13]. Dostupné z: https://blog.heureka.cz/2019/03/21/lecke-9-jak-na-obrazky-a-popisky/.